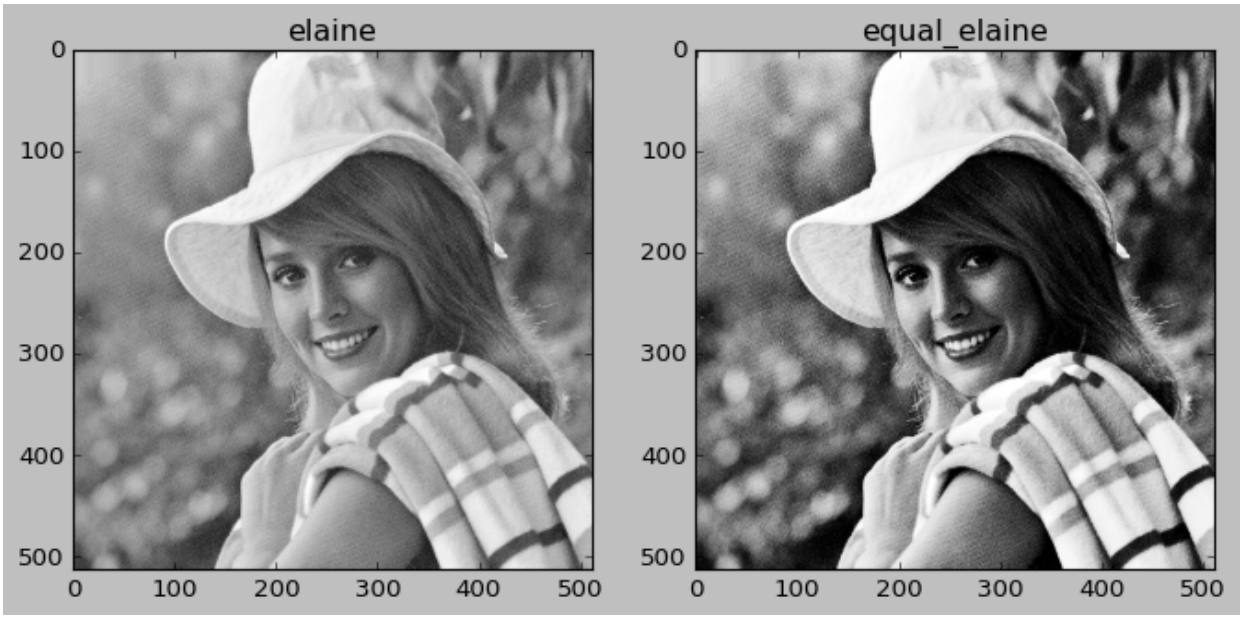
In [ ]:
```
import cv2
import matplotlib.pyplot as plt
from functions import *
import numpy as np
plt.style.use(plt.style.available[5])
```

1.1.1. For two cases as without and with histogram equalization (uniform histogram), display the quantized image in (4, 8, 16, 32, 64, 128) Levels and its histograms. Also, the optimum mean square error obtained for each case. Discuss and report the results for the gray Elaine image. It should be noted, you can use rgb2gray, histeq and immse functions for this problem.

In [ ]:
```
elaine = cv2.imread('Elaine.bmp',cv2.IMREAD_GRAYSCALE)

# equalize elaine image
elaine_histo = calc_hitogram(elaine)
normal_elaine_histo = normalizeHistogram(elaine_histo,elaine.shape[0],elaine.shape[1])
elaine_cdf = calc_cdf(normal_elaine_histo)
equal_elaine = reMap(elaine,elaine_cdf)

# plot original and equalized image
figure = plt.figure(figsize=(10,10))
figure.add_subplot(1,2,1)
plt.imshow(elaine,cmap=plt.gray())
plt.title('elaine',color='black')

figure.add_subplot(1,2,2)
plt.imshow(equal_elaine)
plt.title('equal_elaine',color='black')
```

Out[ ]:
```
Text(0.5, 1.0, 'equal_elaine')
```

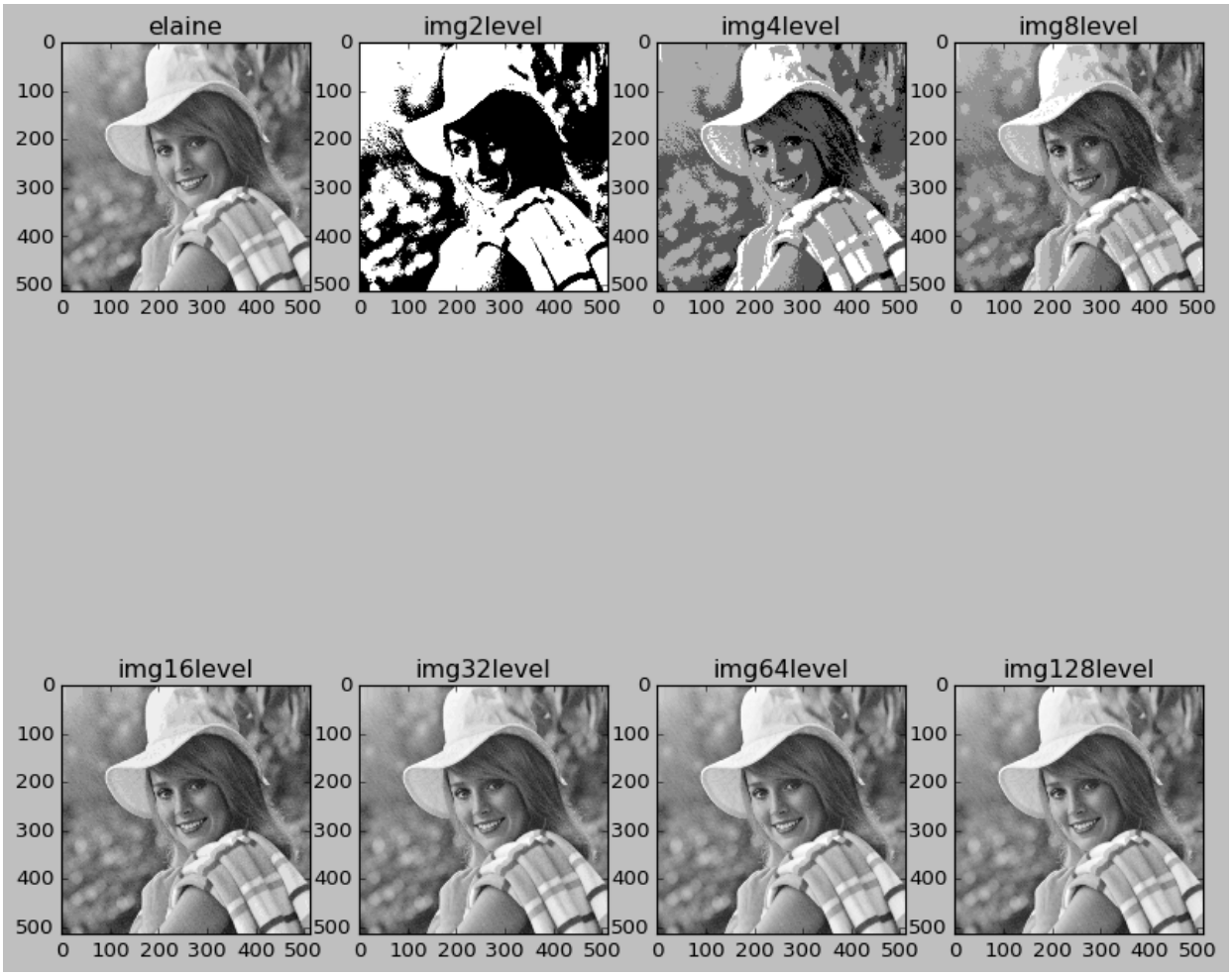


In [ ]:
```
# quantize original image
img2level = quantize_simulation(elaine,1) # 2 level = 2**1
img4level = quantize_simulation(elaine,2) # 4 level = 2**2
img8level = quantize_simulation(elaine,3) # 8 level = 2**3
img16level = quantize_simulation(elaine,4) # 16 level = 2**4
img32level = quantize_simulation(elaine,5) # 32 level = 2**5
```

```
img64level = quantize_simulation(elaine,6) # 64 level = 2**6
img128level = quantize_simulation(elaine,7) # 128 level = 2**7
```

In [ ]:

```
figure = plt.figure(figsize=(12,12))

figure.add_subplot(2,4,1)
plt.imshow(elaine)
plt.title('elaine',color='black')

figure.add_subplot(2,4,2)
plt.imshow(img2level)
plt.title('img2level',color='black')

figure.add_subplot(2,4,3)
plt.imshow(img4level)
plt.title('img4level',color='black')

figure.add_subplot(2,4,4)
plt.imshow(img8level)
plt.title('img8level',color='black')

figure.add_subplot(2,4,5)
plt.imshow(img16level)
plt.title('img16level',color='black')

figure.add_subplot(2,4,6)
plt.imshow(img32level)
plt.title('img32level',color='black')

figure.add_subplot(2,4,7)
plt.imshow(img64level)
plt.title('img64level',color='black')

figure.add_subplot(2,4,8)
plt.imshow(img128level)
plt.title('img128level',color='black')

plt.show()
```



In [ ]:
```
# calculate MSE, for original image cases
mse2level = mean_square_error(elaine,img2level)
print(f'mean_square_error(2) = {mse2level}')
mse4level = mean_square_error(elaine,img4level)
print(f'mean_square_error(4) = {mse4level}')
mse8level = mean_square_error(elaine,img8level)
print(f'mean_square_error(8) = {mse8level}')
mse16level = mean_square_error(elaine,img16level)
print(f'mean_square_error(16) = {mse16level}')
mse32level = mean_square_error(elaine,img32level)
print(f'mean_square_error(32) = {mse32level}')
mse64level = mean_square_error(elaine,img64level)
print(f'mean_square_error(64) = {mse64level}')
mse128level = mean_square_error(elaine,img128level)
print(f'mean_square_error(128) = {mse128level}')
```

```
mean_square_error(2) = 5436.5977
mean_square_error(4) = 1346.5181
mean_square_error(8) = 325.0481
mean_square_error(16) = 77.3164
mean_square_error(32) = 17.3819
mean_square_error(64) = 3.4688
mean_square_error(128) = 0.5093
```

In [ ]:
```
# quantize equalized image
img2level = quantize_simulation(equal_elaine,1) # 4 level = 2**1
img4level = quantize_simulation(equal_elaine,2) # 4 level = 2**2
img8level = quantize_simulation(equal_elaine,3) # 8 level = 2**3
img16level = quantize_simulation(equal_elaine,4) # 16 level = 2**4
```

```
img32level = quantize_simulation(equal_elaine,5) # 32 level = 2**5
img64level = quantize_simulation(equal_elaine,6) # 64 level = 2**6
img128level = quantize_simulation(equal_elaine,7) # 128 level = 2**7
```

In [ ]:

```
figure = plt.figure(figsize=(12,12))

figure.add_subplot(2,4,1)
plt.imshow(equal_elaine,cmap=plt.gray())
plt.title('elaine',color='black')

figure.add_subplot(2,4,2)
plt.imshow(img2level)
plt.title('img2level',color='black')

figure.add_subplot(2,4,3)
plt.imshow(img4level)
plt.title('img4level',color='black')

figure.add_subplot(2,4,4)
plt.imshow(img8level)
plt.title('img8level',color='black')

figure.add_subplot(2,4,5)
plt.imshow(img16level)
plt.title('img16level',color='black')

figure.add_subplot(2,4,6)
plt.imshow(img32level)
plt.title('img32level',color='black')

figure.add_subplot(2,4,7)
plt.imshow(img64level)
plt.title('img64level',color='black')

figure.add_subplot(2,4,8)
plt.imshow(img128level)
plt.title('img128level',color='black')

plt.show()
```

In [ ]:
```
# calculate MSE, for equalized image case
mse2level = mean_square_error(equal_elaine,img2level)
print(f'mean_square_error(2) = {mse2level}')
mse4level = mean_square_error(equal_elaine,img4level)
print(f'mean_square_error(4) = {mse4level}')
mse8level = mean_square_error(equal_elaine,img8level)
print(f'mean_square_error(8) = {mse8level}')
mse16level = mean_square_error(equal_elaine,img16level)
print(f'mean_square_error(16) = {mse16level}')
mse32level = mean_square_error(equal_elaine,img32level)
print(f'mean_square_error(32) = {mse32level}')
mse64level = mean_square_error(equal_elaine,img64level)
print(f'mean_square_error(64) = {mse64level}')
mse128level = mean_square_error(equal_elaine,img128level)
print(f'mean_square_error(128) = {mse128level}')
```

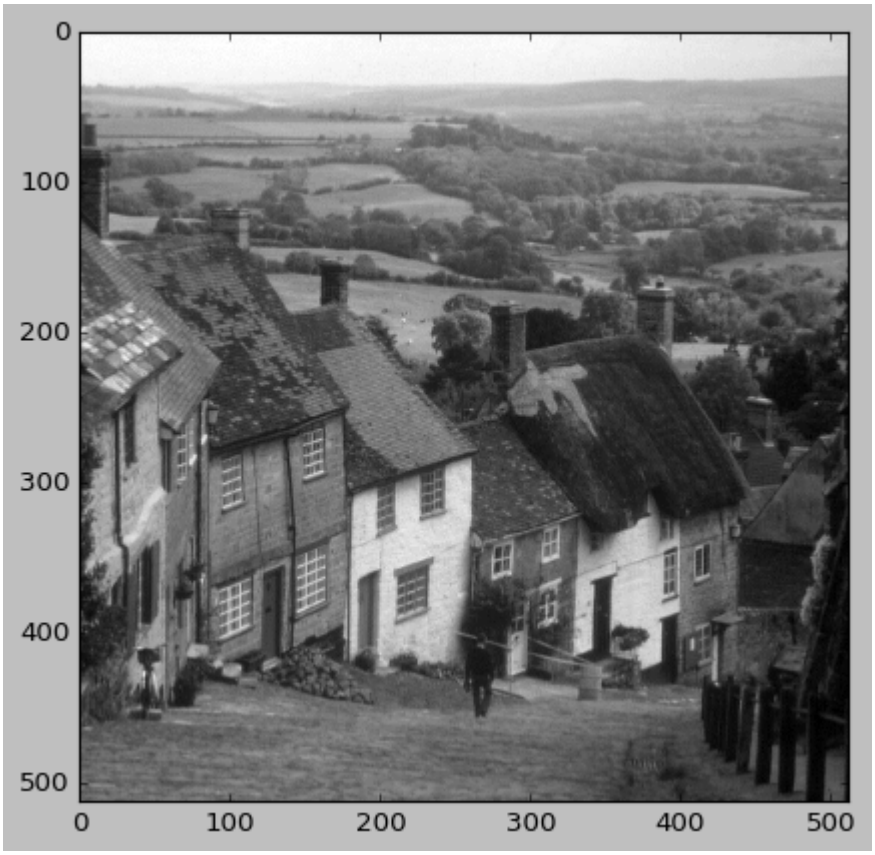
```
mean_square_error(2) = 5296.0716
mean_square_error(4) = 1318.1048
mean_square_error(8) = 325.0791
mean_square_error(16) = 76.2318
mean_square_error(32) = 17.8455
mean_square_error(64) = 3.9539
mean_square_error(128) = 0.5847
```

1.1.2. Write a program which can, firstly, downsample an image by a factor of 2, with and without using the averaging filter, and also, up-sample the previously downsampled images by a factor of 2, using the pixel replication and bilinear interpolation methods, respectively. Display

(zoom of image) and discuss the results obtained with different methods for the Goldhill image. Note, you can use immse function for this problem.

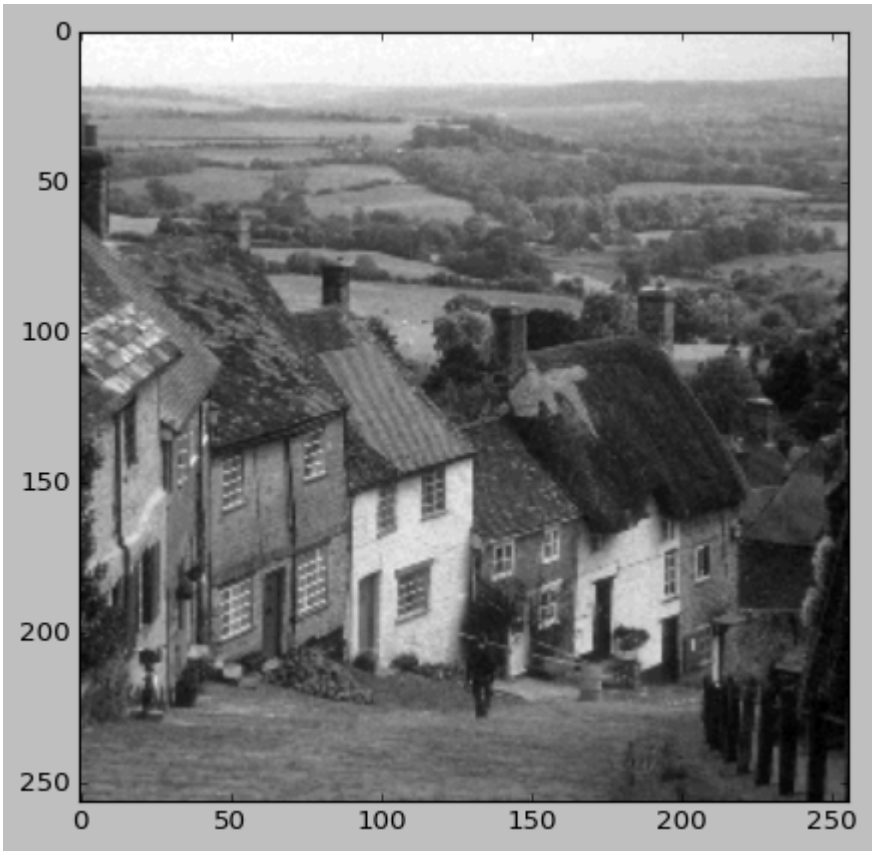
In [ ]:
```
goldhill = cv2.imread('Goldhill.bmp', cv2.IMREAD_GRAYSCALE)
width , length = goldhill.shape
plt.imshow(goldhill,cmap=plt.gray())
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x2630c772dd0>
```



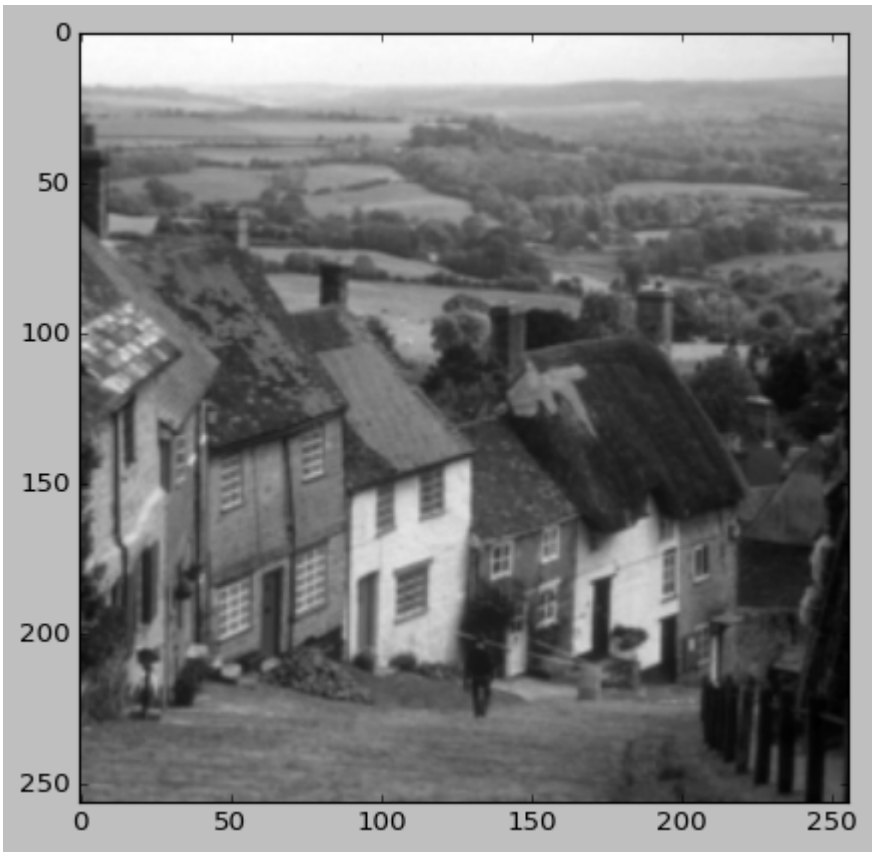
In [ ]:
```
# downsample an image by a factor of 2, without using averaging filter
# we just select rows & columns every other one
downsampled_without_averaging = goldhill[::2,::2]
plt.imshow(downsampled_without_averaging)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x2630c74bf10>
```



In [ ]:
```
# downsample an image by a factor of 2, using averaging filter
downsampled_using_averaging = averaging_filter(goldhill,windowSize=3,downsaplingFactor
plt.imshow(downsampled_using_averaging)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x2630c686c80>
```



In [ ]:
```
# upsample an image by a factor of 2, using replication method
figure = plt.figure(figsize=(10,10))

upsampled_with_replication_1 = replication(downsampled_without_averaging,upsamplngFact
figure.add_subplot(1,2,1)
plt.imshow(upsampled_with_replication_1)
plt.title('upsampled removed rows & columns')

upsampled_with_replication_2 = replication(downsampled_using_averaging,2)
figure.add_subplot(1,2,2)
plt.imshow(upsampled_with_replication_2)
plt.title('upsampled averaged')

plt.show()
```

In [ ]:
```
# upsample an image by a factor of 2, using bilinear interpolation method
figure = plt.figure(figsize=(10,10))

upsampled_with_bilinear_interpolation_1 = np.zeros((downsampled_without_averaging.shap
upsampled_with_bilinear_interpolation1 = interpolate_bilinear(downsampled_without_aver
figure.add_subplot(1,2,1)
plt.imshow(upsampled_with_bilinear_interpolation_1)
plt.title('bilinear_interpolation_1')

upsampled_with_bilinear_interpolation_2 = np.zeros((downsampled_using_averaging.shape[
upsampled_with_bilinear_interpolation2 = interpolate_bilinear(downsampled_using_averag
figure.add_subplot(1,2,2)
plt.imshow(upsampled_with_bilinear_interpolation_2)
plt.title('bilinear_interpolation_2')
```

Out[ ]:
```
Text(0.5, 1.0, 'bilinear_interpolation_2')
```

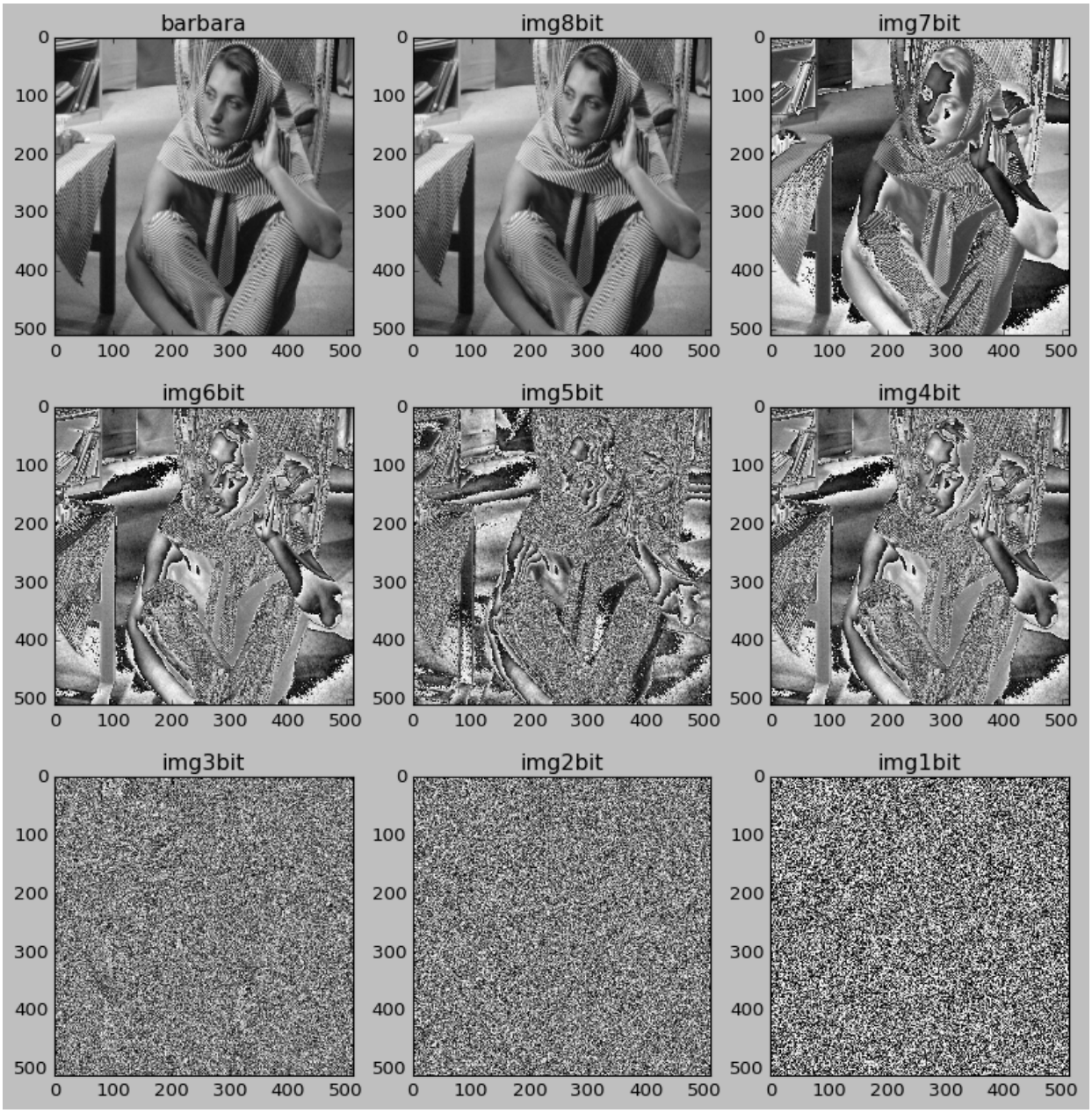
In [ ]:
```
# calculate MSE
removal_down_replication_up = mean_square_error(goldhill,upsampled_with_replication_1)
print(f'(MSE) removal_down_replication_up: {removal_down_replication_up}')

removal_down_bilinear_up = mean_square_error(goldhill,upsampled_with_bilinear_interpol
print(f'(MSE) removal_down_bilinear_up: {removal_down_bilinear_up}')

averaging_down_replication_up = mean_square_error(goldhill,upsampled_with_replication_
print(f'(MSE) averaging_down_replication_up: {averaging_down_replication_up}')

averaging_down_bilinear_up = mean_square_error(goldhill,upsampled_with_bilinear_interp
print(f'(MSE) averaging_down_bilinear_up: {averaging_down_bilinear_up}')
```

```
(MSE) removal_down_replication_up: 133.0759
(MSE) removal_down_bilinear_up: 65.0079
(MSE) averaging_down_replication_up: 99.0850
(MSE) averaging_down_bilinear_up: 142.4705
```

1.1.3. The initial image consists of eight bits of data for each pixel. Create new images using 5, 4, 3, 2 and 1 bit only for each pixel. How many bits are needed to preserve image quality? Does it change from place to place in the image? Discuss about the results. (Test Image Grayscale of Barbara).

In [ ]:
```
barbara = cv2.imread('Barbara.bmp', cv2.IMREAD_GRAYSCALE)

# plt.imshow(barbara,cmap=plt.gray())
img8bit = get_bit_planes(barbara.copy(), 0b11111111)
img7bit = get_bit_planes(barbara.copy(), 0b01111111)
img6bit = get_bit_planes(barbara.copy(), 0b00111111)
img5bit = get_bit_planes(barbara.copy(), 0b00011111)
img4bit = get_bit_planes(barbara.copy(), 0b00001111)
img3bit = get_bit_planes(barbara.copy(), 0b00000111)
img2bit = get_bit_planes(barbara.copy(), 0b00000011)
img1bit = get_bit_planes(barbara.copy(), 0b00000001)


figure = plt.figure(figsize=(12,12))
figure.add_subplot(3,3,1)
plt.imshow(barbara,cmap=plt.gray())
```

```
plt.title('barbara',color='black')

figure.add_subplot(3,3,2)
plt.imshow(img8bit)
plt.title('img8bit',color='black')

figure.add_subplot(3,3,3)
plt.imshow(img7bit)
plt.title('img7bit',color='black')

figure.add_subplot(3,3,4)
plt.imshow(img6bit)
plt.title('img6bit',color='black')

figure.add_subplot(3,3,5)
plt.imshow(img5bit)
plt.title('img5bit',color='black')

figure.add_subplot(3,3,6)
plt.imshow(img6bit)
plt.title('img4bit',color='black')

figure.add_subplot(3,3,7)
plt.imshow(img3bit)
plt.title('img3bit',color='black')

figure.add_subplot(3,3,8)
plt.imshow(img2bit)
plt.title('img2bit',color='black')

figure.add_subplot(3,3,9)
plt.imshow(img1bit)
plt.title('img1bit',color='black')

plt.show()
```


barbara
img8bit
img7bit
img6bit
img5bit
img4bit
img3bit
img2bit
img1bit
0
100
200
300
400
500